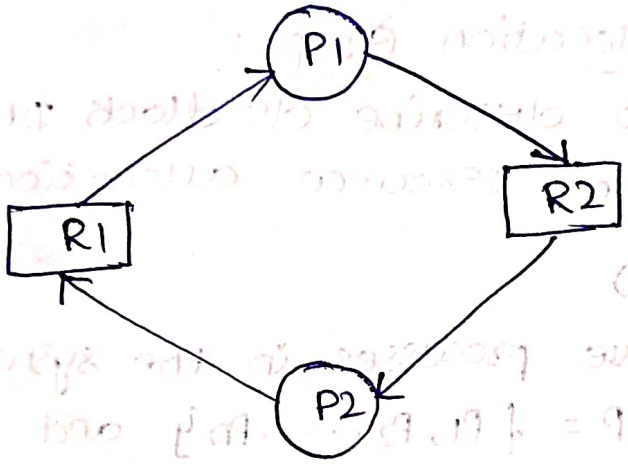
Deadlocks

- A deadlocks is a situation,
A set of processes are said to be in deadlocks state if every process is holding some resources and is waiting for additional resources but those resources are acquired by other processes.



Deadlock characterization

Necessary conditions:

1. Mutual Exclusion
2. Hold and wait
3. No preemption
4. circular wait.

① only one process can use only one resource at a time.

- That means all the resources are non-sharable by default.
- By default it is not possible to share resources among multiple processes as they lead in wrong results.

(2) A process is holding some resources and is waiting for additional resources.

(3) No resource can be forcibly removed from a process holding it until it finishes its execution.

(4) A set of process are said to be in circular wait when P1 holds R1 and requests R2 and process P2 holds R2 while it requests resource R1.

Resource - Allocation Graph:

- Is used to describe deadlock. It is also called system resource allocation graph.

$G = (V, E)$

All the active processes in the system are denoted by $P = \{P_1, P_2 \ldots P_n\}$ and set of resources denoted by $R = \{R_1, R_2 \ldots R_n\}$.

Process is denoted by circle

Resource is denoted by square

Dot within the square represents the no. of instances.

Request edge is an edge from process to resource denoted by $P_i \rightarrow R_j$

Allocated edge or Assignment edge is an edge from resource to process denoted by $R_j \rightarrow P_i$

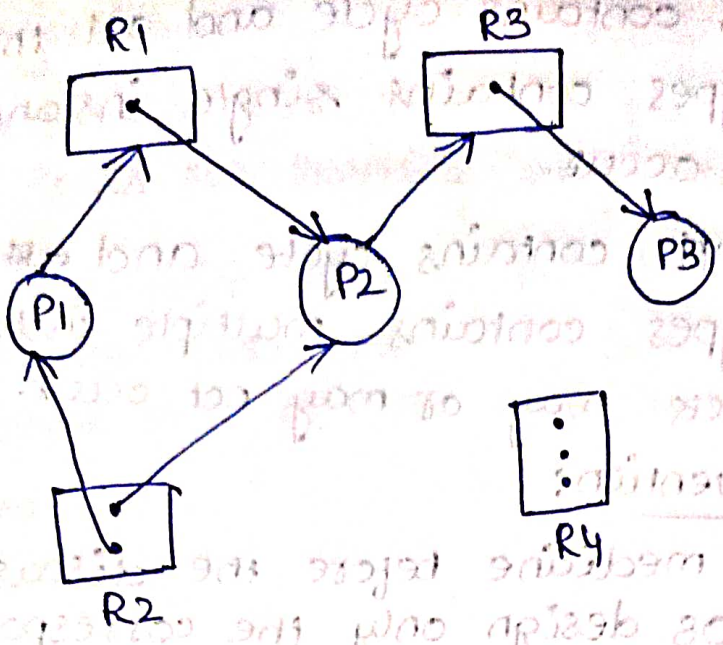
Ex:

$P = \{P_1, P_2, P_3\}$

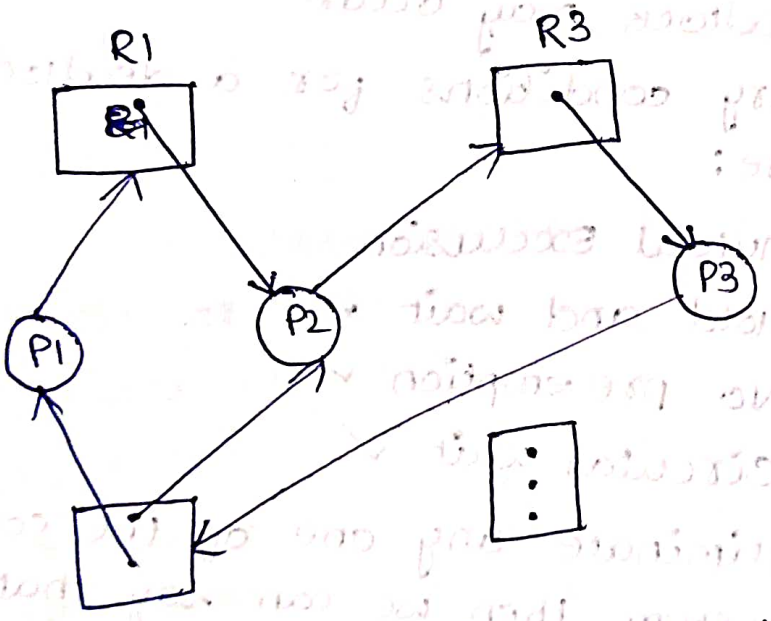
$R = \{R_1, R_2, R_3, R_4\}$

$E = \{P_1 \rightarrow R_1,\ P_2 \rightarrow R_3,\ R_1 \rightarrow P_2,\ R_2 \rightarrow P_2,\ R_2 \rightarrow P_1,$

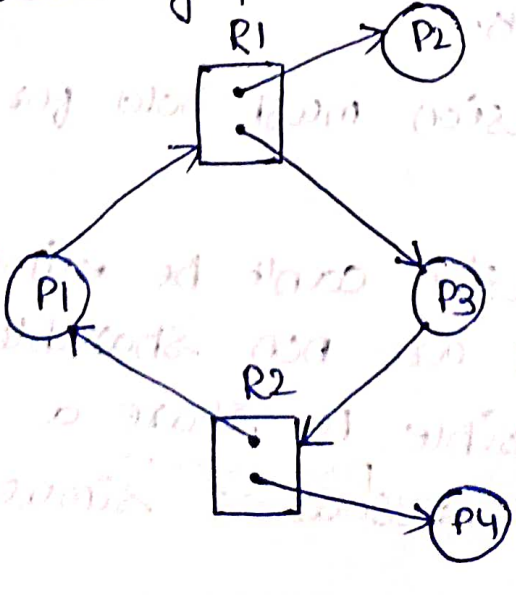
$R_3 \rightarrow P_3\}$



- If the graph contains no cycles, then no process in the system is deadlocks.
- If the graph contain a cycle, then a deadlocks may exist.



Resource allocation graph with a cycle but no deadlock:



- If the graph contains cycle and all the resources types contains single instance then deadlock occur.
- If the graph contains cycle and the resources types contains multiple instances then deadlock may or may not occur.

Deadlocks prevention:

- Taking the medicine before the disease.
- During the OS design only the corresponding OS manufacturers takes necessary precautions so that deadlock may not occur.
- If the 4 conditions occurs simultaneously in the system, then there is a possibility that deadlock may occur.
- Necessary conditions for a deadlock to occur are:
  1. Mutual Exclusion
  2. Hold and wait ✓
  3. No pre emption ✓
  4. Circular wait. ✓
- If we eliminate any one of the condition in the system, then we can say that there is no deadlock.

(1) Mutual Exclusion must hold for non sharable resources.

Mutual exclusion can't be eliminated y bcaz the resources are non sharable by default.

It's not possible to share a resource among multiple ~~resources~~ processes simultaneously.

(2) 2 Approaches:

1. A process can request for a resource only if it is not holding any other resources.

(2). The process has to put the request for all the resources.

- once the process acquires all the resources then only the process has to start its execution.

Disadv's:

1. Poor utilization of resources.

   Ex: A process requires 5 resources and one among is printer which is required at the ending.

2. Starvation.

(3) 1. $P_i \rightarrow R_j \rightarrow P_j$

The OS will checks whether P_i has any acquired resources or not.

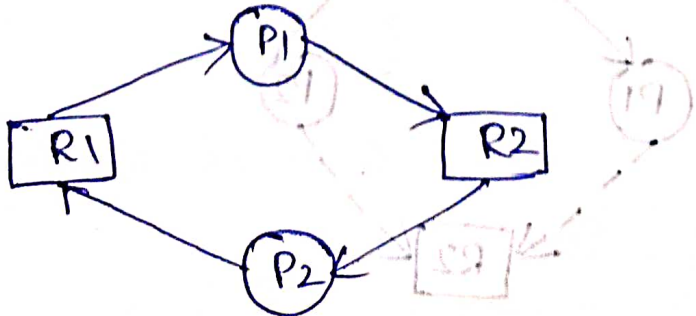
If P_i holds any other resources then OS forcibly releases those resources and allocates those resources to processes which need them. and after some time P_i may request for R_j.

2. $P_i \rightarrow R_j \rightarrow P_j \rightarrow R_k$

   $P_i \rightarrow R_j$ $P_j \rightarrow R_k$

Here, OS is forcibly releasing the resources.

(4)



- The corresponding process has to request for the process in incremental order.

P5 → R0, R3, R2

R2, R3, R6.

Deadlocks avoidance:

1. Resoure- Allocation Graph Alg:
2. Bankers ~~Alloco~~ Alg. - Multiple instances.

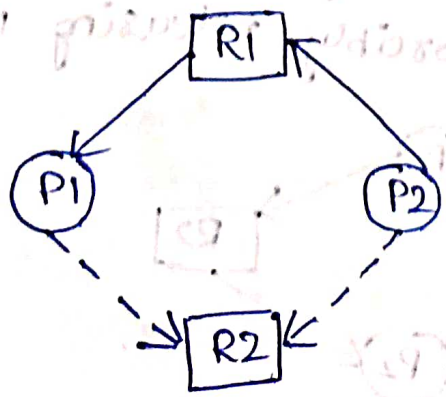
① If the resource type contains single instance, then we use Resource allo. graph alg. for deadlock avoidance.

- Normally request edge and assignment edge is used.
- In addition to the request edge and assignment edge, a new edge called claim edge is added.

1. Claim edge - specifies that a process may request for the resource in the future.
- It is denoted by dotted arrow.

P_i - - -> R_j

This indicates that process P_i may request for resource R_j in the future.



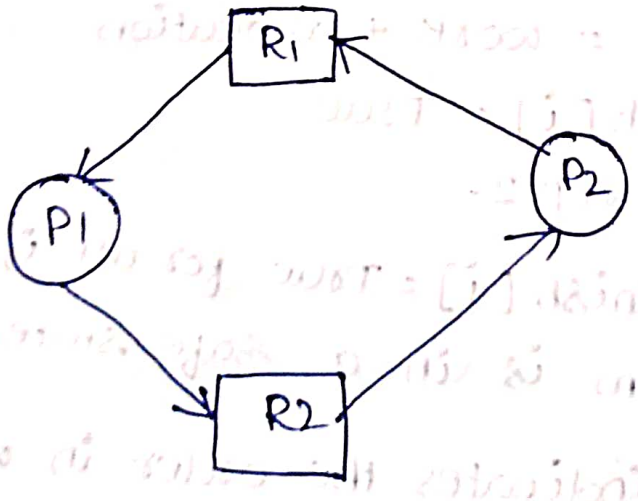
- When process P_i request resource R_j, then the claim edge is converted to a request edge.

$$P_i \longrightarrow R_j$$

- When the resource is free then the OS allocates that resource to the process. Then the request edge is converted to assignment edge.

$$R_j \rightarrow P_i$$

- Once the process execution is over, then the process releases resources, so again that assignment edge is converted to claim edge.
- Cycle detection alg. is used for detecting cycle in the graph.
- If no cycle exists, then the allocation of the resource will leave the system in a safe state.
- If a cycle is found, then the allocation puts the system in an unsafe state.
- Safe state ~~or~~ No deadlocks
- unsafe state - Deadlocks.



Unsafe state in Resource Allocation Graph.

2. Bankers Algorithm: Inorder to avoid deadlo

There are two algs.

1. Safety algorithm
2. Resource - Request alg.

- several data structures must be maintained to implement the bankers alg.
- Let n be the no. of processes in the system and m be the no. of resource types. we need the foll. data structures.

vector → 1. Available - Indicates no of available resource

nxm matrices { 2. Max - defines the max. demand of each proces
3. Allocation - defines the no of resources of each type currently allocated to each proces
4. Need - Indicates the remaining resource need of each process.

1. Safety Alg:

Step1: Work = Available
Finish[i] = False for i=0, 1, 2, 3, 4 . . . . n-1
↳ indicates the corresponding process is not executed till now

Step2: Find an i such that
a. Finish[i] = False
b. Need ≤ Work

Step3: Work = Work + allocation
Finish[i] = True
goto step 2.

Step4: If Finish[i] = True for all i, then the system is in a safe state.

- Safe state indicates the order in which the processes are to be executed and OS will allocate resources to processes in that order only.

Example:

process	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Sol: The need is calculated as

Need = Max - Allocation.

process	Need		
	R1	R2	R3
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

work = Available

work = 3 3 2

Finish

False	False	False	False	False
0	1	2	3	4

P0

Need ≤ Work

7 4 3 ≤ 3 3 2. False

So P0 must wait

P1

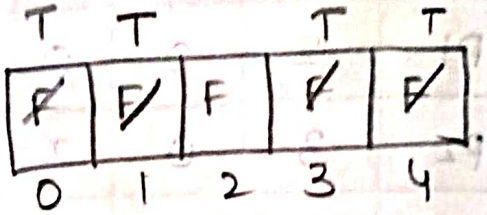
Need ≤ Work

1 2 2 ≤ 3 3 2, True

Work = Work + Allocation

= 3 3 2 + 2 0 0

= 5 3 2



Finish [1] = True

P2

Need ≤ Work

6 0 0 ≤ 5 3 2 False

P2 must wait

P3

0 1 1 ≤ 5 3 2, True

Work = 5 3 2 + 2 1 1

= 7 4 3

P4

Need ≤ Work

4 3 1 ≤ 7 4 3, True

Work = Work + Allocation

= 7 4 3 + 0 0 2

= 7 4 5

Finish [4] = True.

P0

Need ≤ Work

7 4 3 ≤ 7 4 5, True

Work = Work + Allocation

= 7 4 5 + 0 1 0

= 7 5 5

Finish [0] = True

P2

Need ≤ Work

1 2 2 ≤ 7 5 5, True

Work = 7 5 5 + 3 0 2

= 10 5 7

Finish [2] = True.

Safe sequence is < P1, P3, P4, P0, P2 >

2. Resource - Request Alg:

It is an extension to ~~resource~~ safety alg.

Step1: If $Request_i \leq Need_i$ go to Step2.

Step2: If $Request_i \leq Available_i$ goto Step3.

Step3: $Available = Available - Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

If the resulting resource allon. state is safe, the transn. is completed and Pi is allocated its resources.

Ex: Let P1 request for (1, 0, 2)

1. Request ≤ Need

1 0 2 ≤ 1 2 2 True

So goto step 2.

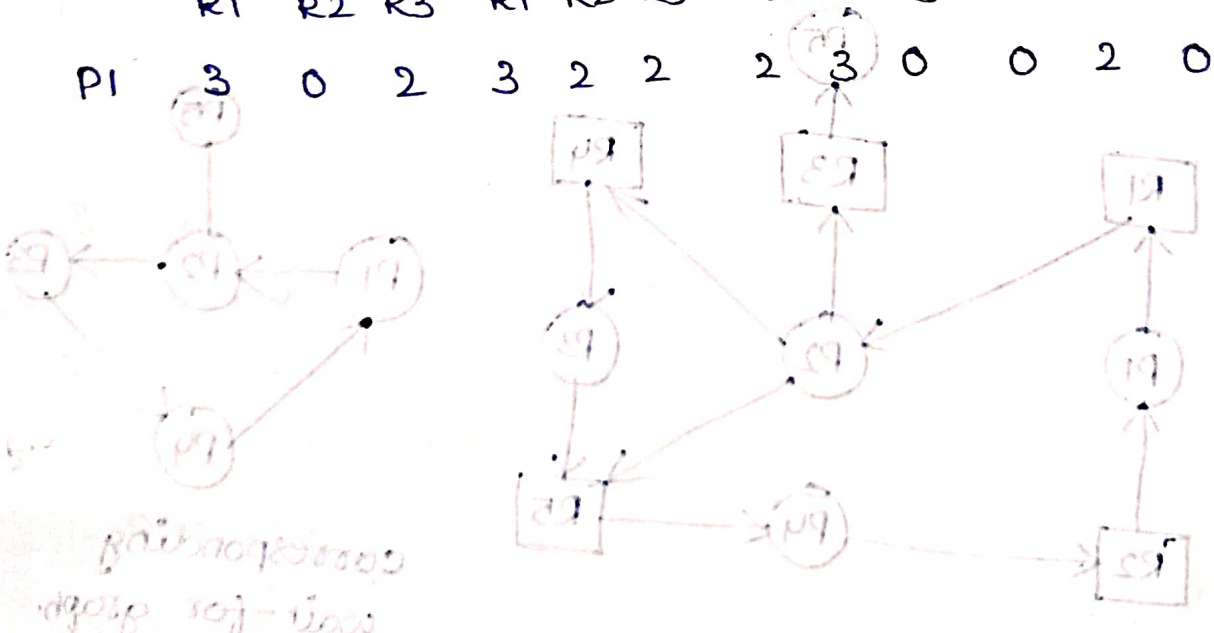
2. Request ≤ Available

1 0 2 ≤ 3 3 2 True

So goto step 3.

3.

	Allocation			Max			Available			Need		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	0	2	3	2	2	2	3	0	0	2	0



eg: $P_1(1,0,2)$

Process	Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3	2	3	0
P_1	3	0	2	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Process	Need		
	R_1	R_2	R_3
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

i) Request$_i \leq$ Need$_i$

1 0 2 $\leq$ 1 2 2 (True)

Request$_i \leq$ Available

1 0 2 $\leq$ 3 3 2 (True)

Available = Available − Request$_i$

= 3 3 2 − 1 0 2

= 2 3 0

Allocation$_i$ = Allocation$_i$ + Request$_i$

= 2 0 0 + 1 0 2

= 3 0 2

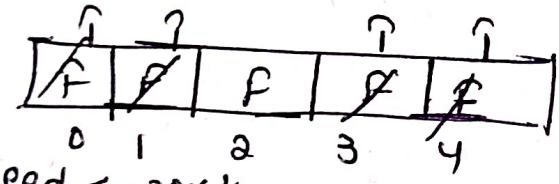
Need$_i$ = Need$_i$ − Request$_i$

= 1 2 2 − 1 0 2

= 0 2 0

work = Available

work = 2 3 0

Finish 

P_0: Need $\leq$ work

7 4 3 $\leq$ 2 3 0 (F)

wait

P_1: Need $\leq$ work.

0 2 0 $\leq$ 2 3 0 (T)

work = work + Allocation

= 2 3 0 + 3 0 2

= 5 3 2.

Finish[1] = true

P_2: Need $\leq$ work

6 0 0 $\leq$ 5 3 2

(False)

P_3: Need $\leq$ work

0 1 1 $\leq$ 5 3 2

True.

work = work + Allocation

= 5 3 2 + 2 1 1

= 7 4 3.

Finish[3] = True

P_4: 4 3 1 $\leq$ 7 4 3

work += Allocation

= 7 4 3 + 0 0 2

= 7 4 5.

Finish[4] = T

P_0: 020 ≤ 745 (T)

$\omega = \omega + A$

$= 745 + 010$

$= 755$

Finish[0] = True

P_2: 600 ≤ 755 (T)

$\omega = \omega + A$

$= 755 + 302$

$= 1057$

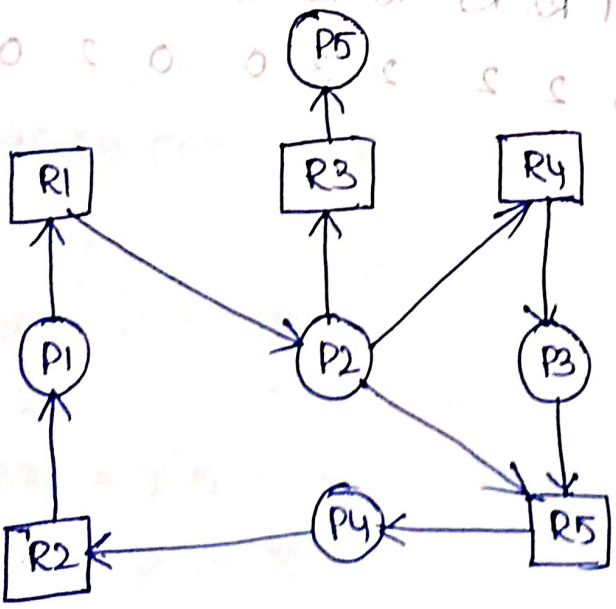
∴ Sequence of safe: P_1 P_3 P_4 P_0 P_2.

Deadlocks detection

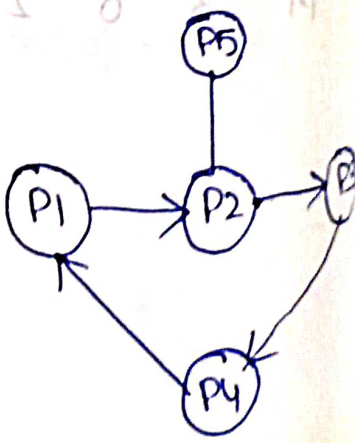
- After allocating resources to the ~~es~~process, The OS will checks whether there is any deadlocks or not.
- Inorder to ~~dead~~detect deadlocks we have two approaches:
  1. ~~Resource Allocation Graph~~ Single instance of each resource type
  2. Several instances of a resource type

1. ~~RAG~~ Single Instance

- If all the resources have only a single instance, then deadlocks detection alg. that uses a variant of RAG called wait-for graph.
- wait-for graph is obtained from resource-allocation graph.
- Nodes of resource is removed and collapsing the appropriate edge i.e assignment edge and request edge.



RAG



corresponding wait-for graph

a. An edge from P_i to P_j in a wait-for graph implies that process P_i is waiting for process P_j to release a resouree that P_i needs.

b. An edge from P_i to P_j exists in a wait-for graph if and only if the corresponding resource-allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_i$ for some resource R_q.

- If the wait for graph contains cycle, then there is a deadlock.
- To detect deadlocks, the system needs to maintain the wait for graph and periodically to invoke an alg. that searches for a cycle in the graph.

2. Several Instances:

Alg:

1. work = available
   Finish[i] = False if allocation $\neq 0$
   Finish[i] = True if allocation $= 0$
2. Find an i such that
   Finish[i] = False
   Request $\leq$ work
3. work = work + allocation
   Finish[i] = True, goto step 2
4. If Finish[i] = False for $i = 0, 1, 2, \ldots n-1$, then the system is in deadlocks state otherwise safe state.

Example:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	7	2	6
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Sol:

work = 0 0 0

Finish

F	F	F	F	F
0	1	2	3	4

P0

Request ≤ work

0 0 0 ≤ 0 0 0, True

work = work + allocation

= 0 0 0 + 0 1 0

= 0 1 0

Finish [0] = True

P1

Request ≤ work

2 0 2 ≤ 0 1 0, False

So, P1 must wait

P2

Request ≤ work

~~0 0 0~~ ≤ 0 1 0, ~~False~~ True

~~So P2 request wait~~

work = work + allocation

= 0 1 0 + 3 0 3

= 3 1 3, Finish[2] = True.

∴ No deadlock detected.

P3

Request ≤ work

1 0 0 ≤ 3 1 3, True

work = work + allocation

= 3 1 3 + 2 1 1

= 5 2 4

Finish [3] = True

P4

Request ≤ work

0 0 2 ≤ 5 2 4, True

work = work + allocation

= 5 2 4 + 0 0 2

= 5 2 6

Finish [4] = True

P1

Request ≤ work

2 0 2 ≤ 5 2 6

work = 5 2 6 + 2 0

= 7 2 6

Finish [1] = True.

Safe Sequence, <P0, P2, P3, P4, P1>

Recovery from deadlock

Two approaches

1. Process Termination
   a. Abort all deadlocked processes
   b. Abort one process at a time until deadlock cycle is eliminated.

2. Resource pre-emption - 3 issues need to be considered
   a. selecting a victim
   b. Rollbacks
   c. Starvation.

① a. 10 processes
3 P's are suffering from deadlock

P0 - 95%.
P1 - 90%.
P2 - 88%.

} To execute these processes till now it required so much of CPU time, memory, I/O devices .... once we kill these processes then all those resources will be wasted. so we can say that this is very expensive approach. In this approach the resources will be wasted unnecessarily.

b. Kill one process at a time and by applying deadlock detection technique inorder to know deadlock cycle is eliminated or not.
- Too much overhead will be there on the computer as we have to apply deadlock detection alg. after killing every process. This is the dis. adv. of this approach.

- While aborting a process we must consider the foll. factors.

* Lowest priority
* Least amount of output produced so far
* Least amount of processor time consumed so far
* Least total resources allocated so far.

2. Releasing resources.
   a. Which resources and which processes are to be pre empted?
   b. Restarting all the processes from the beginning after pre-emption of resources or processes requires Rollback.
   c. There may be a possibility of starvation if a single process resources are released. In that case the process has to wait for longer period of time which leads to starvation.

By considering these three issues the resources have to be pre-empted from the process.

Methods for Handling Deadlocks :

1. Deadlock Prevention
2. Deadlock Avoidance
3. Deadlock Detection
4. Deadlock Recovery.